

## Certificate of Registration



This Certificate issued under the seal of the Copyright Office in accordance with title 17, *United States Code*, attests that registration has been made for the work identified below. The information on this certificate has been made a part of the Copyright Office records.

Acting United States Register of Copyrights and Director

Registration Number

**TX 8-400-276**

Effective Date of Registration:

January 13, 2017

### Title

Title of Work: emoshaGraphics CAD

### Completion/Publication

Year of Completion: 2017

Date of 1st Publication: January 10, 2017

Nation of 1st Publication: United States

### Author

- Author: Louis Arthur Coffelt  
Author Created: computer program  
Citizen of: United States  
Domiciled in: United States  
Year Born: 1959

### Copyright Claimant

Copyright Claimant: Louis Arthur Coffelt  
231 E. Alessandro Blvd., 6A-504, Riverside, CA, 92508, United States

### Rights and Permissions

Name: Louis Arthur Coffelt  
Email: louis.coffelt@gmail.com  
Telephone: (951)790-6086  
Address: 231 E. Alessandro Blvd., 6A-504  
Riverside, CA 92508 United States

### Certification

Name: Louis Arthur Coffelt, Jr.  
Date: January 13, 2017  
Applicant's Tracking Number: 1133emgcadv1

```
#pragma once
#include "C:\\test\\cad_dll\\Convert_Binary_To_Doubles.h"
#include "C:\\test\\cad_dll\\WriteBinaryDoublesOrIntDLL.h"
#include "C:\\test\\cad_dll\\Convert_Doubles_To_Binary.h"
```

```
#include "Drawing_Cls.h"
#include <cmath>
```

TX 8-400-276

```
namespace CppWinForm1
{
```

```
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::IO;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace Convert_Binary_To_Doubles;
    using namespace WriteBinaryDoublesDLLx;
    using namespace Convert_Doubles_To_Binary;
    /// <summary>
    /// Summary for MyForm
    /// </summary>
    public ref class MyForm : public System::Windows::Forms::Form
    {
```

```
    public:
        MyForm(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
        }
```

```
    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~MyForm()
        {
            if (components)
            {
                delete components;
            }
        }
```

```
    private: System::Windows::Forms::PictureBox^ fileButton;
    private: System::Windows::Forms::PictureBox^ newOpenCloseSaveButton;
```

```
    protected:
```



protected:

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container ^components;

private: System::Windows::Forms::PictureBox^ viewButton;

private: System::Windows::Forms::PictureBox^ rotateZoomButton;

private: System::Windows::Forms::PictureBox^ coordSystemImage;

private: System::Windows::Forms::PictureBox^ planeButton;

private: System::Windows::Forms::PictureBox^ triangleButton;

private: System::Windows::Forms::PictureBox^ discButton;

private: System::Windows::Forms::PictureBox^ ringButton;

private: System::Windows::Forms::PictureBox^ cylinderButton;

private: System::Windows::Forms::PictureBox^ sphereButton;

private: System::Windows::Forms::PictureBox^ hemisphereButton;

private: System::Windows::Forms::PictureBox^ helpButton;

private: System::Windows::Forms::PictureBox^ outputImage;

private: System::Windows::Forms::PictureBox^ selectedSurfaceColor;

private: System::Windows::Forms::PictureBox^ deleteSurfaceButton;

private: System::Windows::Forms::PictureBox^ editSurfaceButton;

private: System::Windows::Forms::PictureBox^ colorPalletImage;

private: System::Windows::Forms::TextBox^ xp0box;

private: System::Windows::Forms::TextBox^ yp0box;

private: System::Windows::Forms::TextBox^ zp0box;

private: System::Windows::Forms::TextBox^ zp1box;

private: System::Windows::Forms::TextBox^ yp1box;

private: System::Windows::Forms::TextBox^ xp1box;

private: System::Windows::Forms::TextBox^ radiusBox;

private: System::Windows::Forms::PictureBox^ newSurfaceButton;

private: System::Windows::Forms::PictureBox^ saveSurfaceButton;

private: System::Windows::Forms::TextBox^ surfaceDescriptionBox;

private: System::Windows::Forms::PictureBox^ saveDescriptionButton;

private: System::Windows::Forms::PictureBox^ button188x32;

private: System::Windows::Forms::Label^ SelectedColorTipLabel;

private: System::Windows::Forms::Label^ recentFileName0button;

private: System::Windows::Forms::Label^ recentFileName1button;

private: System::Windows::Forms::Label^ recentFileName2button;

private: System::Windows::Forms::Label^ recentFileName3button;

```

private: System::Windows::Forms::TextBox^ zp2box;
private: System::Windows::Forms::TextBox^ yp2box;
private: System::Windows::Forms::TextBox^ xp2box;
private: System::Windows::Forms::TextBox^ zp3box;
private: System::Windows::Forms::TextBox^ yp3box;
private: System::Windows::Forms::TextBox^ xp3box;
private: System::Windows::Forms::PictureBox^ lineButton;
private: System::Windows::Forms::PictureBox^ filletLinearButton;
private: System::Windows::Forms::PictureBox^ surfaceDensityButton;
private: System::Windows::Forms::PictureBox^ lightSourceButton;
private: System::Windows::Forms::TextBox^ doubleInput2box;
private: System::Windows::Forms::TextBox^ doubleInput1box;
private: System::Windows::Forms::TextBox^ doubleInput0box;
private: System::Windows::Forms::PictureBox^ saveDoublesButton;
private: System::Windows::Forms::Label^ double0Label;
private: System::Windows::Forms::Label^ double1Label;
private: System::Windows::Forms::Label^ double2Label;
private: System::Windows::Forms::PictureBox^ cancelInputButton;

private: System::Windows::Forms::PictureBox^ densityChoiceButton;
private: System::Windows::Forms::Label^ densityValueLabel;
private: System::Windows::Forms::PictureBox^ startPageButton;
private: System::Windows::Forms::PictureBox^ start_page_dwg_ico;
private: System::Windows::Forms::DataGridViewTextBoxColumn^ surface_type;
private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column6;
private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column7;
private: System::Windows::Forms::PictureBox^ cancelSurfaceButton;
private: System::Windows::Forms::PictureBox^ newProjectButton;

private: System::Windows::Forms::DataGridView^ surfaceListBox;

```

#pragma region Windows Form Designer generated code

```

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        System::ComponentModel::ComponentResourceManager^ resources = (gcnew
System::ComponentModel::ComponentResourceManager(MyForm::typeid));
        System::Windows::Forms::DataGridViewCellStyle^ dataGridViewCellStyle7 = (gcnew
System::Windows::Forms::DataGridViewCellStyle());
        System::Windows::Forms::DataGridViewCellStyle^ dataGridViewCellStyle8 = (gcnew
System::Windows::Forms::DataGridViewCellStyle());
        System::Windows::Forms::DataGridViewCellStyle^ dataGridViewCellStyle9 = (gcnew
System::Windows::Forms::DataGridViewCellStyle());
        this->fileButton = (gcnew System::Windows::Forms::PictureBox());
        this->newOpenCloseSaveButton = (gcnew System::Windows::Forms::PictureBox());
        this->viewButton = (gcnew System::Windows::Forms::PictureBox());
        this->rotateZoomButton = (gcnew System::Windows::Forms::PictureBox());
        this->coordSystemImage = (gcnew System::Windows::Forms::PictureBox());
    }

```



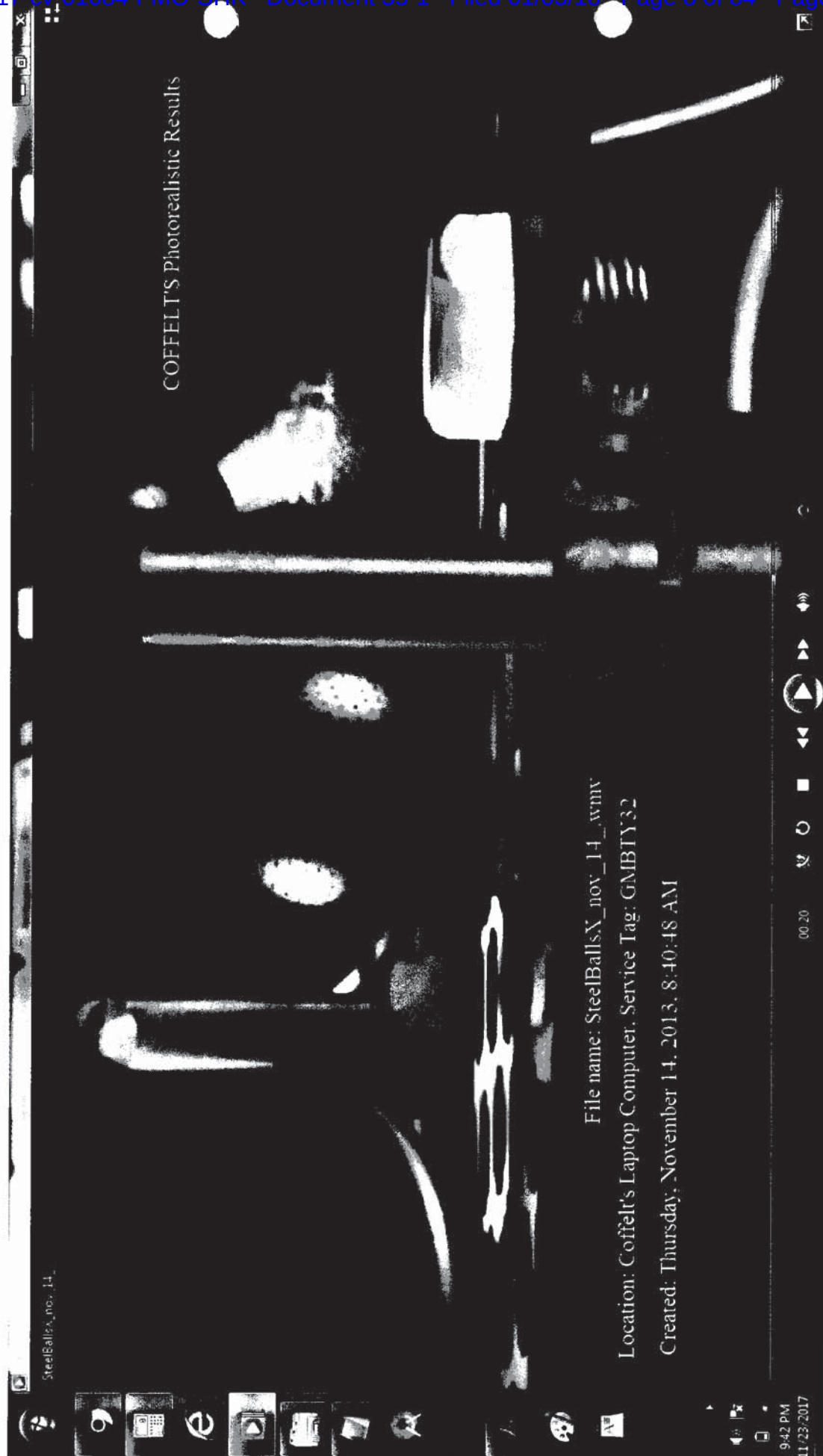
# **EXHIBIT 107**



COFFELT'S CAD WORK 2013



# **EXHIBIT 108**



COFFELT'S CAD WORK 2013

73



# **EXHIBIT 109**



COFFELT'S CAD WORK 2013



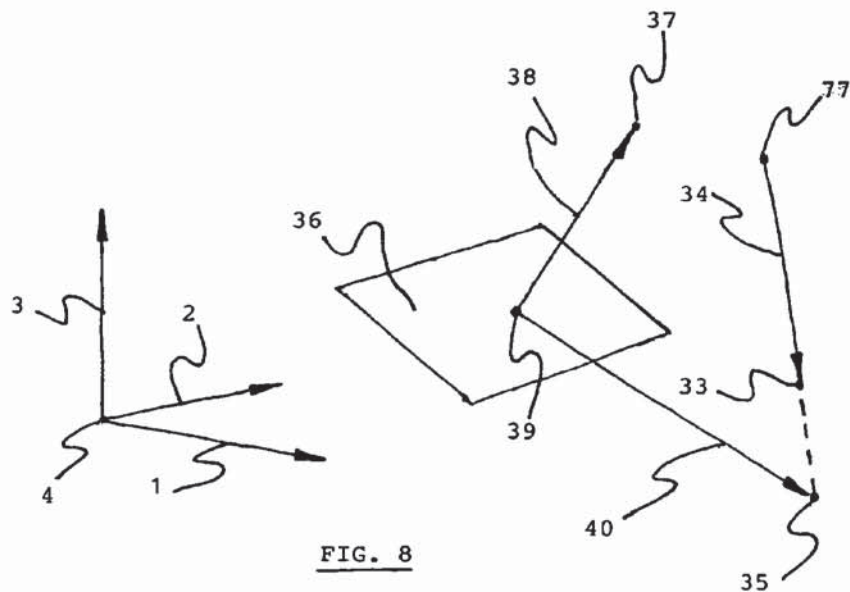
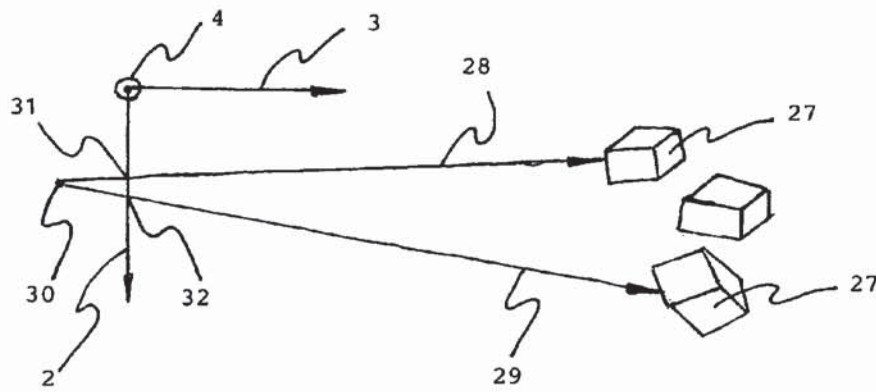
# **EXHIBIT 110**

U.S. Patent

Dec. 24, 2013

Sheet 4 of 4

US 8,614,710 B2



VECTOR PLANE INTERSECTION



## VECTOR PLANE INTERSECTION

US 8,614,710 B2

9

```

columnD=pti*pixelsPerInchD; column=unsigned int(col-
umnD); Indexx=row*bitmapPixelWidth+column;
priorlength=lengthv[Indexx]; currentlength=sqrt
(vppti*vppti+vpptj*vpptj+vpptk*vpptk);
if(currentlength<priorlength) <[lengthv[Indexx]=cur-
rentlength; ]> pti+=0.001; count1++; ]> while(count2<700)
<[ptj=m0*pti-4.3; vppti=vpi-pti; vpptj=vpi-ptj;
vpptk=vpi-ptk; IntersectVectorWithPlane(intpti, intptj,
intptk, vpi, vpj, vpk, pti, ptj, ptk, N1i, N1j, N1k, N0i, N0j,
N0k); if(intpti<0.0 or intpti> bitmapWidthInches or
intptj<0.0 or intptj> bitmapHeightInches)<[pti+=0.001;
count2++; if(count2<700)<[continue; ]> else <[break; ]>>
rowD=ptj*pixelsPerInchD; row=unsigned int(rowD);
columnD=pti* pixelsPerInchD; column=unsigned int(col-
umnD); Indexx=row* bitmapPixelWidth+column;
priorlength=lengthv[Indexx]; currentlength=sqrt
(vppti*vppti+vpptj*vpptj+vpptk*vpptk); difference1=abs
(currentlength-priorlength); if(difference1<0.0001)<
[pointIsVisible=true; red=240; green=0; blue=0;
bitmapx.SetPixel(row, column, red, green, blue); ]> else
<[pointIsVisible=false; ]> pti+=0.001; count2++; ]> while
(count3<900)<[ptj=m1*pti-2.89; vppti=vpi-pti; vpptj=vpi-
ptj; vpptk=vpi-ptk; IntersectVectorWithPlane(intpti, intptj,
intptk, vpi, vpj, vpk, pti, ptj, ptk, N1i, N1j, N1k, N0i, N0j,
N0k); if(intpti<0.0 or intpti> bitmapWidthInches or
intptj<0.0 or intptj> bitmapHeightInches)<[pti+=0.001;
count3++; if(count3<900)<[continue; ]> else<[break; ]>>
rowD=ptj*pixelsPerInchD; row=unsigned int(rowD);
columnD=pti*pixelsPerInchD; column=unsigned int(col-
umnD); Indexx=row*bitmapPixelWidth+column;
priorlength=lengthv[Indexx]; currentlength=sqrt
(vppti*vppti+vpptj*vpptj+vpptk*vpptk); difference1=abs
(currentlength-priorlength); if(difference1<0.0001)<
[pointIsVisible=true; red=0; green=0; blue=250;
bitmapx.SetPixel(row, column, red, green, blue); ]> else
<[pointIsVisible=false; ]> pti+=0.001; count3++; ]>>

```

FIG. 8 shows a perspective view of a vector (34) intersecting a plane (36) at point (35). A normal vector (38) of plane (36) is shown. The normal vector is formed by point N0 (39) and point N1 (37). Point N (39) is on plane (36). Vector (34) is formed of any two points, point (77) and point (33). Vector (40) is formed of two points, point (35) and point (39). Vector (40) is in plane (36). The following c++ code shows an example of 'Vector Plane Intersection', and is the function used above. where, intpti, intptj, intptk is intersection point (35); pt1 is point (77); pt0 is point (33); N0 is point (39); N1 is point (37):

```

c++ <[void IntersectVectorWithPlane(double& intptiP,
double& intptjP, double& intptkP, double pt1i, double pt1j,
double pt1k, double pt0i, double pt0j, double pt0k, double
N1iP, double N1jP, double N1kP, double N0iP, double N0jP,
double N0kP)<[double Ni=N1iP-N0iP; double Nj=N1jP-
N0jP; double Nk=N1kP-N0kP; double testdenom=abs(pt1i-
pt0i); if(testdenom<1.0e-9)<[return; ]> double mji=(pt1j-
pt0j)/(pt1i-pt0i); double mki=(pt1k-pt0k)/(pt1i-pt0i);
testdenom=abs((Ni+Nj*mji+Nk*mki); if(testdenom<1.0e-9)
<[return; ]> tempi=(N0iP*Ni+NjP*mji*pt0i-NjP*pt0j+
NjP*N0jP+NkP*mki*pt0i-NkP*pt0k+NkP*N0kP)/(Ni+Nj*mji+
NkP*mki); intptiP=tempi; intptjP=mji*(tempi-pt0i)+pt0j;
intptkP=mki*(tempi-pt0i)+pt0k; ]>>

```

The general formula for intptiP is formed by combining a general formula for a plane with a general formula for a line. More specifically, the projection of a vector to the i-j plane, and the projection of the vector to the i-k plane. The general formula for a plane is set in an equation that any vector in the plane dotted with the planes normal vector is zero. N dot v is zero. where N is the normal vector of the plane, and v is any

10

vector in the plane. A general formula for a line is:  $j = mji \cdot (i - i0) + j0$  where  $i0$ ,  $j0$ , and  $mji$  are given. Also,  $k = mki \cdot (i - i0) + k0$  where  $i0$ ,  $k0$ , and  $mki$  are given. These two equations for a line are substituted in the equation for a plane; and solved for  $i$ . This substitution eliminates variables  $j$  and  $k$ . Only  $i$  remains in the equation. Solving for  $i$  yields the equation above in the c++ code snippet.

The graphic object structure analysis may also include reflection vectors. A reflection vector can be derived for any point in a graphic object. This reflection vector may intersect any graphic object. e.g. any plane, sphere, or surface. The intersection point can be assigned any selected pixel color. For example, a light source contacts a particular point on a blue surface; a reflection vector is calculated at this particular point; the reflection vector intersects a red sphere; the intersection point is set to blue. The following c++ code snippet sets forth an example to calculate a reflection vector.

```

c++ <[void ReflectionVector(double& rpti, double & rptj,
double& rptk, double Ni, double Nj, double Nk, double s1i,
double s1j, double s1k, double ai, double aj, double ak)<
[double si=s1i-ai; double sj=s1j-aj; double sk=s1k-ak;
double lengths=sqrt(si*si+sj*sj+sk*sk); double
lengthN=sqrt(Ni*Ni+Nj*Nj+Nk*Nk); double NdotS=
(Ni*si+Nj*sj+Nk*sk)/(lengthN*lengths); double
testDot=abs(NdotS); if(testDot<1.0e-6)<[return; ]> else
if(testDot>0.9999)<[rpti=s1i; rptj=s1j; rptk=s1k; return; ]>
double phi=a cos(NdotS); double Nri=0.0; double Nrj=0.0;
double Nrkk=0.0; double rxi=0.0; double rxj=0.0; double
rxk=0.0; double tlen=2.0*lengths*sin(phi); Nri=Nj*sk-
sj*Nk; Nrj=-(Ni*sk-si*Nk); Nrkk=Ni*sj-si*Nj;
rxi=Nj*Nrk-Nrj*Nk; rxj=-(Ni*Nrk-Nri*Nk);
rxk=Ni*Nrj-Nri*Nj; double lengthrx=sqrt(rxi*rxi+rxj*rxj+
rxk*rxk); if(lengthrx>1.0e-6)<[cx=tlen/lengthrx; ]> else
<[return; ]> double tri=cx*rxi; double trj=cx*rxj; double
trk=cx*rxk; rpti=s1i+tri; rptj=s1j+trj; rptk=s1k+trk; ]>>

```

In the above example for the reflection vector, The tail end of the reflection vector is ai, aj, ak; the terminal end of the reflection vector is rpti, rptj, rptk. The reflection vector is:  $(rpti-ai)i + (rptj-aj)j + (rptk-ak)k$ ; ai, aj, ak, the intersection point of a light source vector with the plane. Ni, Nj, Nk is the normal vector of the plane.

A graphic object structure may also include translucent surfaces. A translucent surface can be derived by intersecting a vector with one or more surfaces. Next, set the selected pixel color at a relatively less pixel per inch resolution. For example, for a bitmap having a resolution of 1000 pixels per inch, a translucent surface can be attained by setting a background image at about 500 pixels per inch. For example, a foreground rectangular translucent blue surface; and a background linear red surface; initially all pixels in the bitmap are blue; Next, the program iterates thru the equation of the line; and assign a red pixel at a density of 500 pixels per inch.

One method to create a translucent surface is using a 'next least length' concept. This concept is related to a 'visible' point on the geometric object. Opaque surfaces described above, have only one point per steradian which is 'visible'. In comparison, for a translucent surface, there may be to or three or more points in one particular steradian which are 'visible'. e.g. the visible points are 2 points having the least position vector length. The following is an example c++ code snippet showing a method to calculate a translucent surface:

The following is a summary of c++ code for translucent surface calculation: This code may be set in line with the above c++ code examples; c++ <vector> priorLength0, <vector> priorLength1, and <vector> priorLength2 contain values having a respective next least length; for example, at index 33, priorLength0[33]=22.15; priorLength1[33]=28.76; prior-

# **EXHIBIT 111**



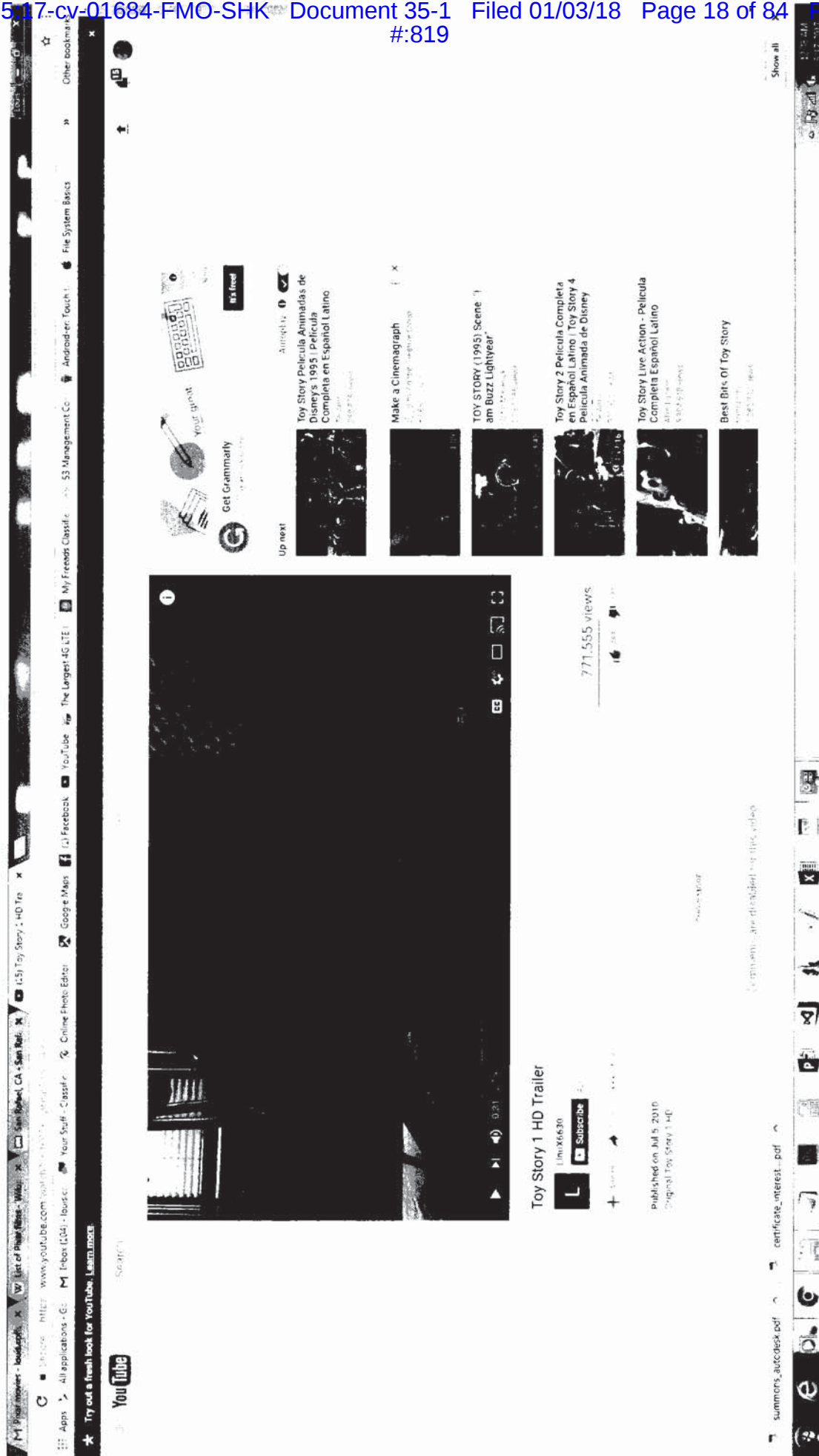
# STATE OF ART GRADIENTS year 1970 through 2010

80



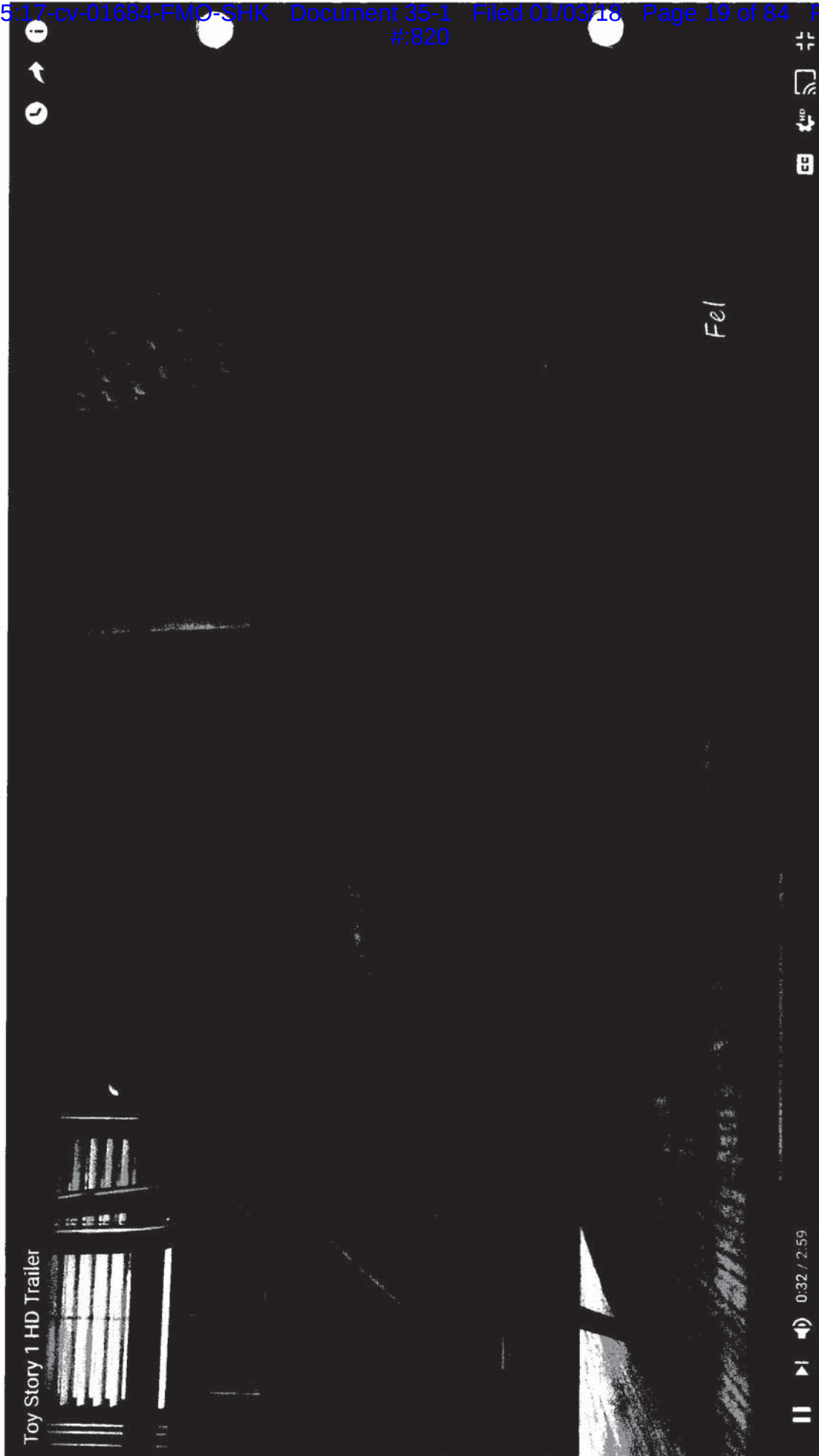


## **EXHIBIT 112**



PIXAR RESULTS 1995

83



PIXAR RESULTS 1995

84

## **EXHIBIT 113**





86



A Bug's Life (beginning)

2:02 / 2:43

PIXAR RESULTS 1998

## **EXHIBIT 114**



★ You have received a YouTube notification. Try out a fresh look for YouTube. Learn more.

Apps > All applications - G > Mail > Inbox (1,141) - Iouis.co > Your Stuff - Classif > Online Photo Editor > Google Maps > Facebook > YouTube > The Largest 4G LTE > My Freebies Classic > 53 Management Co > Android-on: Touch > File System Basics > Other bookmarks > Log out

YouTube

Toy Story 2 - Official Trailer #2 [1999]

3,366,201 views

Published on May 24, 2009

\*\*\*\*\*PLEASE SUBSCRIBE\*\*\*\*\* TO MY WEEKLY SPECIAL MOVIE TRAILER SERIES\*\*\*\*\*

Comments: 244

summers\_autodesk.pdf certificate\_interest...pdf

Toy Story 2 - Official Trailer #2 [1999]

Up next

Toy Story 3: Trailer

Make a Cinemagraph

Toy Story 2 Bloopers

Toy Story 2 Pelicula Completa en Español Latino | Toy Story 4 Pelicula Animada de Disney

Live Action Toy Story

Toy Story 4 Trailer #1 - June 16 2019

TOY STORY (1995) Scene 1

PIXAR RESULTS 1999

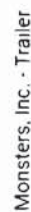




PIXAR RESULTS 1999

90

## **EXHIBIT 115**

[illegible]

Published on Jan 24, 2013

2000

92



PIXAR RESULTS 2001

## **EXHIBIT 116**



YouTube interface showing a video player for "Toy Story 3: Trailer" with a view count of 8,061,065 views. The video player includes a progress bar and a play button. Below the video player, there are several suggested videos, including "Toy Story 3: Best Moments - Cute Buzz & Jessie Moments", "Toy Story 3: Woody Memorable Moments", "Toy Story 4 Trailer #1 - June 16 2019", "Toy story 3 Woody, Jesse, and Buzz vs the enemy", and "Toy Story 3 - Best Scenes". The interface also shows a search bar, a navigation menu, and a list of recommended videos.

PIXAR RESULTS 2010



Toy Story 3: Trailer



0:23 / 2:18

PIXAR RESULTS 2010

96

## **EXHIBIT 117**

YouTube

Try out a fresh look for YouTube. Learn more.

pixar cars 2 trailer

UP NEXT

Cars 3 - Official US Trailer  
Disney Pixar  
15:26  
15,326,893 views

Cars 3 'Rivalry' Official Trailer  
Disney Pixar  
1:27  
1,234,567 views

Time Travel Mater Short Films  
Disney Pixar  
1:00  
1,234,567 views

CARS 3 ALL TRAILERS - 2017  
Pixar Animation  
1:14  
1,234,567 views

Cars 3 DELETED SCENES & Alternate Endings  
Disney Pixar  
1:00  
1,234,567 views

Car 2 (2011) - Best Scenes  
Disney Pixar  
1:00  
1,234,567 views

Cars 2 - Theatrical Trailer  
Disney Pixar  
1:00  
1,234,567 views

cars 2 - Trailer 2  
Disney-Pixar  
1:00  
20,215,800 views

Published on Nov 15, 2010  
Order here: <http://disney.com>  
Now available on Blu-ray™, DVD, and Pixar Store.

summons\_autodesk.pdf  
certificate\_interest...pdf

98

PIXAR RESULTS 2011



PIXAR RESULTS 2011



## **EXHIBIT 118**

/00

YouTube

Try out a fresh look for YouTube. Learn more.

pixar monsters university trailer

Monsters University - Best Scenes

11,605,415 views

Published on Jan 17, 2017

ALL THE COME-TOUS OF THIS VIDEO IS OWNED BY WALT DISNEY PICTURES

Subscribe to watch more videos from Disney or other cool video channels

Comments

summers\_autodesk.pdf certificate\_interest.pdf

101

MONSTERS UNIVERSITY

Up next

Groupon

Finding Dory - Dory Memorable Moments

How Older Men Regain Glory

Zootopia - Best Scenes

Touching Goodbye Scene - Monsters Inc. (Boo & Kitty)

Monsters University Mike Best Moments (HD) - Monsters University Clips Best Moments

Moana - Ridiculous moments

MINIONS GIANT in Millions

PIXAR RESULTS 2013



PIXAR RESULTS 2013

## **EXHIBIT 119**



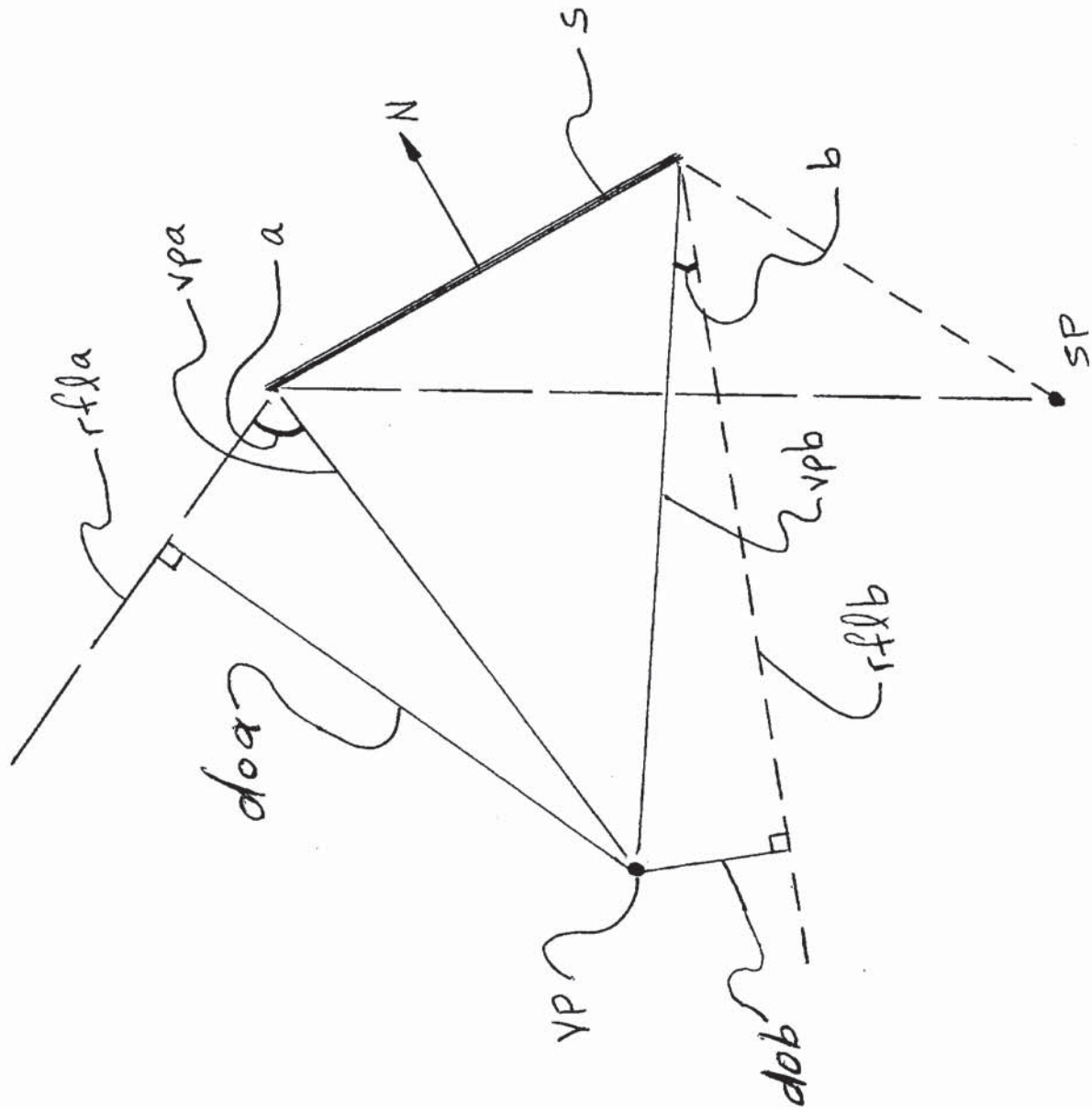
104



Cars 3 - Official US Trailer

PIXAR RESULTS 2017

## **EXHIBIT 120**

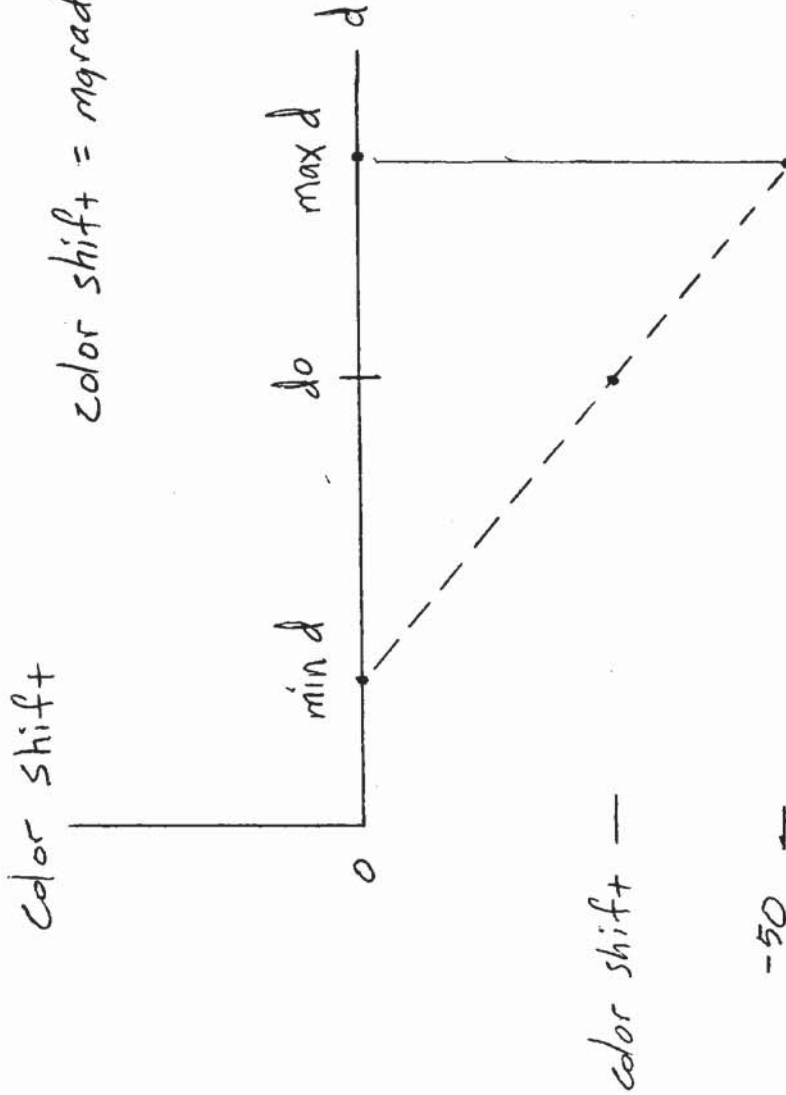


COFFELT GRADIENT WORK



$$mgrad = \frac{-50}{(maxd - mind)}$$

$$color\ shift = mgrad (do - mind)$$



COFFELT GRADIENT WORK

# **EXHIBIT 121**

**STATEMENT OF FACTS RE:  
OSL COPYRIGHT INFRINGEMENT**

The following OSL computer program is an infringement of Coffelt's Gradient Work 2010, U.S. application No. 1-5121154211.

The following OSL computer program is an infringement of Coffelt's Photorealistic Gradient Work, U.S. application No. 1-5376971191.

The following OSL computer program is an infringement of Coffelt's Gradient Work, Registration No. TXu002049564.

The location of OSL infringing source code is identified with reference (A) through (J) in this exhibit.

The basis for OSL is a derivative of Coffelt's source code (Complaint at paragraph 66) is in the last pages of this exhibit.

On Tuesday, August 01, 2017, 6:16:42 PM Coffelt downloaded a copy of OSL source code from:

<https://github.com/imageworks/OpenShadingLanguage>      **(Infringing URL)**

The OSL Infringing Code is as follows:

110

**OSL file name:**

OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp at line 174:

```

(A) virtual float sample(const OSL::ShaderGlobals& sg, float rx, float ry, float rz, OSL::Dual2<OSL::Vec3>& wi, float& pdf) const
{
    float cosNO = -N.dot(sg.I);
    if (cosNO > 0)
    {
        // reflect the view vector
        Vec3 R = (2 * cosNO) * N + sg.I;
        TangentFrame tf(R);

        float phi = 2 * float(M_PI) * rx; (C)
        float sp, cp;
        OIIO::fast_sincos(phi, &sp, &cp);
        float cosTheta = OIIO::fast_safe_pow(ry, 1 / (exponent + 1));
        float sinTheta2 = 1 - cosTheta * cosTheta;
        float sinTheta = sinTheta2 > 0 ? sqrtf(sinTheta2) : 0;
        wi = tf.get(cp * sinTheta, sp * sinTheta, cosTheta);
        // leave derivs 0?
        float cosNI = N.dot(wi.val());
        if (cosNI > 0)
        {
            pdf = (exponent + 1) * float(M_1_PI / 2) * OIIO::fast_safe_pow(cosTheta, exponent);
            return cosNI * (exponent + 2) / (exponent + 1);
        }
    }
    return pdf = 0;
}

```



**OSL file name:**

OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp at line 550:

```

Vec3 sampleMicronormal(const Vec3 wo, float randu, float randv) const
{
    /* Project wo and stretch by alpha values */
    Vec3 swo = wo;
    swo.x *= xalpha;
    swo.y *= yalpha;
    swo = swo.normalize();
    // figure out angles for the incoming vector

    float cos_theta = std::max(swo.z, 0.0f);

    float cos_phi = 1;
    float sin_phi = 0;
    /* Normal incidence special case gets phi 0 */
    if (cos_theta < 0.99999f)
    {
        float invnorm = 1 / sqrtf(SQR(swo.x) + SQR(swo.y));
        cos_phi = swo.x * invnorm;
        sin_phi = swo.y * invnorm;
    }

    Vec2 slope = Distribution::sampleSlope(cos_theta, randu, randv);

    /* Rotate and unstretch slopes */
    Vec2 s(cos_phi * slope.x - sin_phi * slope.y, sin_phi * slope.x + cos_phi * slope.y);
    s.x *= xalpha;
    s.y *= yalpha;
    float mlen = sqrtf(s.x * s.x + s.y * s.y + 1);
    Vec3 m(fabsf(s.x) < mlen ? -s.x / mlen : 1.0f, fabsf(s.y) < mlen ? -s.y / mlen : 1.0f, 1.0f / mlen);
    return m;
}

```

**OSL file name:**

OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.h at line 50:

```

Color3 sample(const ShaderGlobals& sg, float rx, float ry, float rz, Dual2<Vec3>& wi, float& pdf) const
{
    float accum = 0;
    for (int i = 0; i < num_bsdfs; i++)
    {
        if (rx < (pdfs[i] + accum))
        {
            rx = (rx - accum) / pdfs[i];

            rx = std::min(rx, 0.999999994f);

            // keep result in [0,1)
            Color3 result = weights[i] * (bsdfs[i]->sample(sg, rx, ry, rz, wi, pdf) / pdfs[i]);
            pdf *= pdfs[i];
            // we sampled PDF i, now figure out how much the other bsdfs contribute to the chosen direction
            for (int j = 0; j < num_bsdfs; j++)
            {
                if (i == j) continue;
                float bsdf_pdf = 0;
                Color3 bsdf_weight = weights[j] * bsdfs[j]->eval(sg, wi.val(), bsdf_pdf);
                MIS::update_eval(&result, &pdf, bsdf_weight, bsdf_pdf, pdfs[j]);
            }
            return result;
        }
        accum += pdfs[i];
    }
}

```

(F)

113

**OSL file name:**

OpenShadingLanguage-master\src\testrender\testrender.cpp at line 418

```
Color3 subpixel_radiance(float x, float y, Sampler& sampler, ShadingContext* ctx)
{
```

```
    Ray r = camera.get(x, y);
    Color3 path_weight(1, 1, 1);
```

```
    Color3 path_radiance(0, 0, 0); (I)
```

```
    int prev_id = -1;
    float bsdf_pdf = std::numeric_limits<float>::infinity();
    // camera ray has only one possible direction
    bool flip = false;
    for (int b = 0; b <= max_bounces; b++)
    {
```

```
        // trace the ray against the scene
        Dual2<float> t; int id = prev_id;
        if (!scene.intersect(r, t, id))
```

```
    {
        // we hit nothing? check background shader
        if (backgroundShaderID >= 0)
```

```
    {
        if (backgroundResolution > 0)
        {
```

```
            float bg_pdf = 0;
```

```
            Vec3 bg = background.eval(r.d.val(), bg_pdf); (G)
```

```
            path_radiance += path_weight * bg * MIS::power_heuristic<MIS::WEIGHT_WEIGHT>(bsdf_pdf, bg_pdf);
```

```
        }
```

**(H) (J)**

114

## FACTS AND BASIS OSL IS A DERIVATIVE OF COFFELT'S WORK

(A)

**Coffelt:** 0000 rflx = rptx - ptx00a;  
0001 rfly = rpty - pty00a;  
0002 rflz = rptz - ptz00a;

**OSL:** OSL::Dual2<OSL::Vec3>& wi

**Basis:** (i) Coffelt's source code ("rflx, rfly, rflz") in the Complaint at paragraph 66 lines 0000 through 0002 are components of the reflection vector based on the surface normal N and the view direction.

(ii) ("wi") is type ("Vec3") and therefore is a vector having 3 components x, y, and z (c++ standard variable).

(iii) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(iv) Comment in OSL file ("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line 178 ("// reflect the view vector") confirms code is directed to the reflection vector and direction of view.

(v) The location of this OSL source code ("OSL::Dual2<OSL::Vec3>& wi") is file name:  
("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line: 175.

(vi) This OSL source code ("OSL::Dual2<OSL::Vec3>& wi") is available to the public at:  
<https://github.com/imageworks/OpenShadingLanguage>

For these reasons, This OSL source code ("OSL::Dual2<OSL::Vec3>& wi") is equivalent to Coffelt's source code in the Complaint at paragraph 66 lines 0000 through 0002.

115



(B)

0003 lenrfl = sqrt(rflx \* rflx + rfly \* rfly + rflz \* rflz);  
Coffelt: 0004 vpdotrfl = (vpax \* rflx + vpay \* rfly + vpaz \* rflz) / (lenvpa \* lenrfl);

OSL: float cosNO = -N.dot(sg.I);

**Basis:** (i) dot product is the cosine of angle between 2 vectors See EXHIBIT 124.

- (ii) Coffelt's source code ("vpdotrfl") in the Complaint at paragraph 66, line 0004 is the dot product of the view vector and reflection vector. ("lenrfl") the length of the reflection vector; is a standard component of a vector dot product equation.
- (iii) ("sg.I") inherently contains the length of a vector. The length of a vector is a required component of a vector dot product. See EXHIBIT 124 ("magnitude of the vector").

(iv) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(v) Comment in OSL file ("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line 178 ("// reflect the view vector") confirms OSL source code ("float cosNO = -N.dot(sg.I)") is directed to the dot product between the reflection vector and direction of view.

(vi) The location of this OSL source code ("float cosNO = -N.dot(sg.I)") is file name:  
("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line: 176.

(vii) This OSL source code ("float cosNO = -N.dot(sg.I)") is available to the public at:  
<https://github.com/imageworks/OpenShadingLanguage>

For these reasons, the OSL source code ("float cosNO = -N.dot(sg.I)") is equivalent to Coffelt's source code in the Complaint at paragraph 66, line 0003 and line 0004.

116



(C)

**Coffelt:** 0005 theta = acos(vpdotrfl);

**OSL:** float phi = 2 \* float(M\_PI) \* rx;

**Basis:** (i) Coffelt's source code ("theta") in the Complaint at paragraph 66, line 0005 is the angle between the view vector and reflection vector.

(ii) OSL source code ("float phi = 2 \* float(M\_PI) \* rx") is the angle between the view vector and reflection vector.

(iii) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(iv) Comment in OSL file ("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line 178 ("// reflect the view vector") confirms code is directed to the reflection vector and direction of view.

(v) The location of this OSL source code ("float phi = 2 \* float(M\_PI) \* rx") is file name:  
("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line: 181.

(vi) This OSL source code ("float phi = 2 \* float(M\_PI) \* rx") is available to the public at:  
<https://github.com/imageworks/OpenShadingLanguage>

For these reasons, the OSL source code ("float phi = 2 \* float(M\_PI) \* rx") is equivalent to Coffelt's source code in the Complaint at paragraph 66, line 0005.

117

(D)

**Coffelt:** 0006 mgrad = -50 / (max\_d - min\_d);

**OSL:** Vec2 slope = Distribution::sampleSlope(cos\_theta, randu, randv);

**Basis:** (i) Coffelt's source code ("mgrad") in the Complaint at paragraph 66, line 0006 is the slope of a line between the maximum distance and the minimum distance to the view point. See EXHIBIT 120.

(ii) OSL source code ("Vec2 slope = Distribution::sampleSlope(cos\_theta, randu, randv)") is the slope of a line between the maximum distance and the minimum distance to the view point.

(iii) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(iv) The term ("Distribution::sampleSlope") confirms that the slope is based on the distribution containing the maximum distance and the minimum distance to the view point. e.g. the equation uses ("cos\_theta") the angle between the direction of view and the reflection vector.

(v) Comment in OSL file ("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line 178 ("// reflect the view vector") confirms code is directed to the reflection vector and direction of view.

(vi) The location of this OSL source code ("Vec2 slope = Distribution::sampleSlope(cos\_theta, randu, randv)") is file name:  
("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line: 569.

(vii) This OSL source code ("Vec2 slope = Distribution::sampleSlope(cos\_theta, randu, randv)") is available to the public at:  
<https://github.com/imageworks/OpenShadingLanguage>

For these reasons, the OSL source code ("Vec2 slope = Distribution::sampleSlope(cos\_theta, randu, randv)") is equivalent to Coffelt's source code in the Complaint at paragraph 66, line 0006.

118

(E)

**Coffelt:** 0006 mgrad = -50 / (max\_d - min\_d);

**OSL:** float cos\_theta = std::max(swo.z, 0.0f);

**Basis:** (i) Coffelt's source code ("max\_d") in the Complaint at paragraph 66, line 0006 is the maximum distance between the view vector and reflection vector.

(ii) OSL source code ("float cos\_theta = std::max(swo.z, 0.0f)") is the maximum distance between the view vector and reflection vector.

(iii) ("std::max") is a standard C++ function which derives a maximum value of 2 values. See EXHIBIT 125.

(iv) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(v) Comment in OSL file ("OpenShadingLanguage-master\src\testrender\shading.cpp") at line 178 ("// reflect the view vector") confirms code is directed to the reflection vector and direction of view.

(vi) The location of this OSL source code ("float cos\_theta = std::max(swo.z, 0.0f)") is file name: ("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line: 558.

(vii) This OSL source code ("float cos\_theta = std::max(swo.z, 0.0f)") is available to the public at: <https://github.com/imageworks/OpenShadingLanguage>

For these reasons, the OSL source code ("float cos\_theta = std::max(swo.z, 0.0f)") is equivalent to Coffelt's source code in the Complaint at paragraph 66, line 0006.

119



(F)

**Coffelt:** 0006 mgrad = -50 / (max\_d - min\_d);

**OSL:** rx = std::min(rx, 0.99999994f);

**Basis:** (i) Coffelt's source code ("min\_d") in the Complaint at paragraph 66, line 0006 is the minimum distance between the view vector and reflection vector.

(ii) OSL source code ("rx = std::min(rx, 0.99999994f)") is the minimum distance between the view vector and reflection vector.

(iii) ("std::min") is a standard C++ function which derives a minimum value of 2 values. See EXHIBIT 126.

(iv) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(v) Comment in OSL file ("OpenShadingLanguage-master\src\testrender\shading.cpp") at line 178 ("// reflect the view vector") confirms code is directed to the reflection vector and direction of view.

(vi) The location of this OSL source code ("rx = std::min(rx, 0.99999994f)") is file name:

("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.h") at line: 55.

(vii) This OSL source code ("rx = std::min(rx, 0.99999994f)") is available to the public at:  
<https://github.com/imageworks/OpenShadingLanguage>

For these reasons, the OSL source code ("rx = std::min(rx, 0.99999994f)") is equivalent to Coffelt's source code in the Complaint at paragraph 66, line 0006.

120

(G)

**Coffelt:** 0007 d0 = lenvpa \* sin(theta);

**OSL:** Vec3 bg = background.eval(r.d.val(), bg\_pdf);

**Basis:** (i) Coffelt's source code ("d0 ") in the Complaint at paragraph 66, line 0007 is the distance from the view point to the reflection vector.

(ii) OSL source code ("r.d") is the distance from the view point to the reflection vector.

(iii) The parameter (" $\text{MIS::WEIGHT\_WEIGHT}$ ") in file name: ("OpenShadingLanguage-master\src\testrender\testrender.cpp") at line: 434, confirms ("r.d") is the distance from the view point to the reflection vector.

(iv) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(v) Comment in OSL file ("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line 178 ("// reflect the view vector") confirms code is directed to the reflection vector and direction of view.

(vi) The location of this OSL source code ("r.d") is file name:  
("OpenShadingLanguage-master\src\testrender\testrender.cpp") at line: 433.

(vii) This OSL source code ("r.d") is available to the public at:  
<https://github.com/imageworks/OpenShadingLanguage>

For these reasons, the OSL source code ("r.d") is equivalent to Coffelt's source code in the Complaint at paragraph 66, line 0007.



(H)

**Coffelt:** 0008    `shiftd = mgrad * (d0 - min_d);`

**OSL:**    `path_weight * bg`

**Basis:** (i) Coffelt's source code ("`shiftd` ") in the Complaint at paragraph 66, line 0008 is the numerical quantity of addition to the base color of the surface.

(ii) OSL source code ("`path_weight * bg`") is the numerical quantity of addition to the base color of the surface.

(iii) The parameter ("`<MIS::WEIGHT_WEIGHT>`") in file name: ("`OpenShadingLanguage-master\src\testrender\testrender.cpp`") at line: 434, confirms ("`path_weight * bg`") is the numerical quantity of addition to the base color of the surface.

(iv) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(v) Comment in OSL file ("`OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp`") at line 178 ("`// reflect the view vector`") confirms code is directed to the reflection vector and direction of view.

(vi) The location of this OSL source code ("`path_weight * bg`") is file name:  
("`OpenShadingLanguage-master\src\testrender\testrender.cpp`") at line: 434.

(vii) This OSL source code ("`path_weight * bg`") is available to the public at:  
<https://github.com/imageworks/OpenShadingLanguage>

For these reasons, the OSL source code ("`path_weight * bg`") is equivalent to Coffelt's source code in the Complaint at paragraph 66, line 0008.

122

(D)

**Coffelt:** 0009 blueD = 100.0;  
0010 greenD = 255.0;  
0011 redD = 100.0;

**OSL:** Color3 path\_radiance(0, 0, 0);

**Basis:** (i) Coffelt's source code ("blueD, greenD, redD ") in the Complaint at paragraph 66, line 0009 through line 0011 is declaration of the initial surface color integers.

(ii) OSL source code ("Color3 path\_radiance(0, 0, 0)") is declaration of the initial surface color integers.

(iii) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(iv) Comment in OSL file ("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line 178 ("// reflect the view vector") confirms code is directed to the reflection vector and direction of view.

(v) The location of this OSL source code ("Color3 path\_radiance(0, 0, 0)") is file name:  
("OpenShadingLanguage-master\src\testrender\testrender.cpp") at line: 421.

(vi) This OSL source code ("Color3 path\_radiance(0, 0, 0)") is available to the public at:  
<https://github.com/imageworks/OpenShadingLanguage>

For these reasons, the OSL source code ("Color3 path\_radiance(0, 0, 0)") is equivalent to Coffelt's source code in the Complaint at paragraph 66, line 0009 through line 0011.

123

(J)

**Coffelt:** 0012 blueD += shiftD;  
0013 greenD += shiftD;  
0014 redD += shiftD;

**OSL:** path\_radiance += path\_weight \* bg

**Basis:** (i) Coffelt's source code ("blueD +=, greenD +=, redD += ") in the Complaint at paragraph 66, line 0012 through line 0014 is shifting the base color of the surface by the numerical quantity ("shiftD").

(ii) OSL source code ("path\_radiance += path\_weight \* bg") is shifting the base color of the surface by the numerical quantity ("path\_weight \* bg").

(iii) OSL source code ("path\_radiance") is type ("Color3") having 3 components, Red, Green, Blue. See file name: ("OpenShadingLanguage-master\src\testrender\testrender.cpp") at line: 421.

(iv) Sony Imageworks publication confirms OSL is based on direction of view and reflection vector See EXHIBIT 123.

(v) Comment in OSL file ("OpenShadingLanguage-master\OpenShadingLanguage-master\src\testrender\shading.cpp") at line 178 ("// reflect the view vector") confirms code is directed to the reflection vector and direction of view.

(vi) The location of this OSL source code ("path\_radiance += path\_weight \* bg") is file name:

("OpenShadingLanguage-master\src\testrender\testrender.cpp") at line: 434.

(vii) This OSL source code ("path\_radiance += path\_weight \* bg") is available to the public at:

<https://github.com/imageworks/OpenShadingLanguage>

For these reasons, the OSL source code ("path\_radiance += path\_weight \* bg") is equivalent to Coffelt's source code in the Complaint at paragraph 66, line 0012 through line 0014.

124



For all of the above reasons, and those in the Complaint, the above identified OSL source code in items (A) through (J) is an unauthorized derivative work based on Coffelt's copyrighted Gradient Work, U.S. Registration No. TXu002049564, Date of registration: On June 12, 2017, year created: 2013

For all of the above reasons, and those in the Complaint, the above identified OSL source code in items (A) through (J) is an unauthorized derivative work based on Coffelt's copyrighted Photorealistic Gradient Work, U.S. Application No.: 1-5376971191, filed: On June 12, 2017, year created: 2013

Coffelt's Gradient Work is a derivative work based on Coffelt's Gradient Work 2010.

Coffelt's Photorealistic Gradient Work is a derivative work based on Coffelt's Gradient Work 2010.

For all of the above reasons, and those in the Complaint, the above identified OSL source code in items (A) through (J) is an unauthorized derivative work based on Coffelt's copyrighted Gradient Work 2010, U.S. Application No.: 1-5121154211, filed: May 13, 2017, year created: 2010

# **EXHIBIT 122**



11/23/2017

imageworks/OpenShadingLanguage: Advanced shading language for production GI renderers

Your account has been flagged.

Because of that, your profile is hidden from the public. If you believe this is a mistake, contact support to have your account status reviewed.

## imageworks / OpenShadingLanguage

### Advanced shading language for production GI renderers

#osl #shading-language #shaders #c-plus-plus #computer-graphics #computer-language #llvm

1,997 commits

14 branches

82 releases

29 contributors

BSD-3-Clause

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

lgritz Trim verbose build output of clutter

Latest commit 61bb74e 13 days ago

.github

Minor formatting revision of PR and Issue templates

site

Fixup for SPI to put the libs in our usual place

src

Trim verbose build output of clutter

testsuite

ctransform full derivative support.

.gitignore

testsuite overhaul -- all temp files end up in build, runtest.py call...

.travis.yml

Bump LLVM minimum from 3.5 to 3.9. (#806)

CHANGES.md

Bump LLVM minimum from 3.5 to 3.9. (#806)

CMakeLists.txt

Simplify the PugiXML logic. (#809)

CONTRIBUTING.md

Docs for PR, Issue, Contributing

INSTALL.md

Bump LLVM minimum from 3.5 to 3.9. (#806)

LICENSE

Docs

Makefile

Simplify the PugiXML logic. (#809)

https://github.com/imageworks/OpenShadingLanguage

127

11/23/2017

imageworks/OpenShadingLanguage: Advanced shading language for production GI renderers

README.md

CHANGES

24 days ago

README.md



# Open Shading Language

Build status:

build passing

## Table of contents

- Introduction
- How OSL is different
- What OSL consists of
- Where OSL has been used
- Building OSL
- Contacts, Links, and References
- Credits

## Introduction

Welcome to Open Shading Language!

<https://github.com/imageworks/OpenShadingLanguage>

128

Open Shading Language (OSL) is a small but rich language for programmable shading in advanced renderers and other applications, ideal for describing materials, lights, displacement, and pattern generation.

OSL was originally developed by Sony Pictures Imageworks for use in its in-house renderer used for feature film animation and visual effects, released as open source so it could be used by other visual effects and animation studios and rendering software vendors. Now it's the de facto standard shading language for VFX and animated features, used across the industry in many commercial and studio-proprietary renderers. Because of this, the work on OSL received an Academy Award for Technical Achievement in 2017.

OSL is robust and production-proven, and has been used in films as diverse as "The Amazing Spider-Man," "Hotel Transylvania," "Edge of Tomorrow," "Ant Man", "Finding Dory," and many more. OSL support is in most leading renderers used for high-end VFX and animation work. For a full list of films and products, see the filmography.

The OSL code is distributed under the "New BSD" license, and the documentation under the Creative Commons Attribution 3.0 Unported License. In short, you are free to use OSL in your own applications, whether they are free or commercial, open or proprietary, as well as to modify the OSL code and documentation as you desire, provided that you retain the original copyright notices as described in the license.

## How OSL is different

OSL has syntax similar to C, as well as other shading languages. However, it is specifically designed for advanced rendering algorithms and has features such as radiance closures, BSDFs, and deferred ray tracing as first-class concepts.

OSL has several unique characteristics not found in other shading languages (certainly not all together). Here are some things you will find are different in OSL compared to other languages:

- Surface and volume shaders compute radiance closures, not final colors.

OSL's surface and volume shaders compute an explicit symbolic description, called a "closure", of the way a surface or volume scatters light, in units of radiance. These radiance closures may be evaluated in particular directions, sampled to find important directions, or saved for later evaluation and re-evaluation. This new approach is ideal for a physically-based renderer that supports ray tracing and global illumination.



In contrast, other shading languages usually compute just a surface color as visible from a particular direction. These old shaders are "black boxes" that a renderer can do little with but execute to find this one piece of information (for example, there is no effective way to discover from them which directions are important to sample). Furthermore, the physical units of lights and surfaces are often underspecified, making it very difficult to ensure that shaders are behaving in a physically correct manner.

- Surface and volume shaders do not loop over lights or shoot rays.

There are no "light loops" or explicitly traced illumination rays in OSL surface shaders. Instead, surface shaders compute a radiance closure describing how the surface scatters light, and a part of the renderer called an "integrator" evaluates the closures for a particular set of light sources and determines in which directions rays should be traced. Effects that would ordinarily require explicit ray tracing, such as reflection and refraction, are simply part of the radiance closure and look like any other BSDF.

Advantages of this approach include that integration and sampling may be batched or re-ordered to increase ray coherence; a "ray budget" can be allocated to optimally sample the BSDF; the closures may be used by for bidirectional ray tracing or Metropolis light transport; and the closures may be rapidly re-evaluated with new lighting without having to re-run the shaders.

- Surface and light shaders are the same thing.

OSL does not have a separate kind of shader for light sources. Lights are simply surfaces that are emissive, and all lights are area lights.

- Transparency is just another kind of illumination.

You don't need to explicitly set transparency/opacity variables in the shader. Transparency is just another way for light to interact with a surface, and is included in the main radiance closure computed by a surface shader.

- Renderer outputs (AOV's) may be specified using "light path expressions."

Sometimes it is desirable to output images containing individual lighting components such as specular, diffuse, reflection, individual lights, etc. In other languages, this is usually accomplished by adding a plethora of "output variables" to the shaders that collect these individual quantities.

11/23/2017

imageworks/OpenShadingLanguage: Advanced shading language for production GI renderers

OSL shaders need not be cluttered with any code or output variables to accomplish this. Instead, there is a regular-expression-based notation for describing which light paths should contribute to which outputs. This is all done on the renderer side (though supported by the OSL implementation). If you desire a new output, there is no need to modify the shaders at all; you only need to tell the renderer the new light path expression.

- Shaders are organized into networks.

OSL shaders are not monolithic, but rather can be organized into networks of shaders (sometimes called a shader group, graph, or DAG), with named outputs of some nodes being connected to named inputs of other nodes within the network. These connections may be done dynamically at render time, and do not affect compilation of individual shader nodes. Furthermore, the individual nodes are evaluated lazily, only when their outputs are "pulled" from the later nodes that depend on them (shader writers may remain blissfully unaware of these details, and write shaders as if everything is evaluated normally).

- Arbitrary derivatives without grids or extra shading points.

In OSL, you can take derivatives of any computed quantity in a shader, and use arbitrary quantities as texture coordinates and expect correct filtering. This does not require that shaded points be arranged in a rectangular grid, or have any particular connectivity, or that any "extra points" be shaded. This is because derivatives are not computed by finite differences with neighboring points, but rather by "automatic differentiation", computing partial differentials for the variables that lead to derivatives, without any intervention required by the shader writer.

- OSL optimizes aggressively at render time

OSL uses the LLVM compiler framework to translate shader networks into machine code on the fly (just in time, or "JIT"), and in the process heavily optimizes shaders and networks with full knowledge of the shader parameters and other runtime values that could not have been known when the shaders were compiled from source code. As a result, we are seeing our OSL shading networks execute 25% faster than the equivalent shaders hand-crafted in C! (That's how our old shaders worked in our renderer.)

## What OSL consists of

The OSL open source distribution consists of the following components:

131



- oslc, a standalone compiler that translates OSL source code into an assembly-like intermediate code (in the form of .oso files).
- liboslc, a library that implements the OSLCompiler class, which contains the guts of the shader compiler, in case anybody needs to embed it into other applications and does not desire for the compiler to be a separate executable.
- liboslquery, a library that implements the OSLQuery class, which allows applications to query information about compiled shaders, including a full list of its parameters, their types, and any metadata associated with them.
- oslinfo, a command-line program that uses liboslquery to print to the console all the relevant information about a shader and its parameters.
- liboslexec, a library that implements the ShadingSystem class, which allows compiled shaders to be executed within an application. Currently, it uses LLVM to JIT compile the shader bytecode to x86 instructions.
- testshade, a program that lets you execute a shader (or connected shader network) on a rectangular array of points, and save any of its outputs as images. This allows for verification of shaders (and the shading system) without needing to be integrated into a fully functional renderer, and is the basis for most of our test suite verification. Along with testrender, testshade is a good example of how to call the OSL libraries.
- testrender, a tiny ray-tracing renderer that uses OSL for shading. Features are very minimal (only spheres are permitted at this time) and there has been no attention to performance, but it demonstrates how the OSL libraries may be integrated into a working renderer, what interfaces the renderer needs to supply, and how the BSDFs/radiance closures should be evaluated and integrated (including with multiple importance sampling).
- A few sample shaders.
- Documentation -- at this point consisting of the OSL language specification (useful for shader writers), but in the future will have detailed documentation about how to integrate the OSL libraries into renderers.

## Where OSL has been used

132

*This list only contains films or products whose OSL use is stated or can be inferred from public sources, or that we've been told is ok to list here. If an OSL-using project is missing and it's not a secret, just email the OSL project leader or submit a PR with edits to this file.*

Renderers and other tools with OSL support (in approximate order of adding OSL support):

- Sony Pictures Imageworks: in-house "Arnold" renderer
- Blender/Cycles
- Chaos Group: V-Ray
- Pixar: PhotoRealistic RenderMan RIS
- Isotropix: Clarisse
- Autodesk Beast
- Appleseed
- Animal Logic: Glimpse renderer
- Image Engine: Gaffer (for expressions and deformers)
- DNA Research: 3Delight
- Ubisoft motion picture group's proprietary renderer
- Autodesk/SolidAngle: Arnold

Films using OSL (grouped by year of release date):

- (2012) Men in Black 3, The Amazing Spider-Man, Hotel Transylvania
- (2013) Oz the Great and Powerful, Smurfs 2, Cloudy With a Chance of Meatballs 2
- (2014) The Amazing Spider-Man 2, Blended, Edge of Tomorrow, 22 Jump Street, Guardians of the Galaxy, Fury, The Hunger Games: Mockingjay - Part 1, Exodus: Gods and Kings, The Interview
- (2015) American Sniper, Insurgent, Avengers Age of Ultron, Ant Man, Pixels, Mission Impossible: Rogue Nation, Hotel Transylvania 2, Bridge of Spies, James Bond: Spectre, The Hunger Games: Mockingjay - Part 2, Concussion
- (2016) Allegiant, Batman vs Superman: Dawn of Justice, The Huntsman, Angry Birds Movie, Alice Through the Looking Glass, Captain America: Civil War, Finding Dory, Piper, Independence Day: Resurgence, Ghostbusters, Star Trek Beyond, Suicide Squad, Kingsglave: Final Fantasy XV, Storks, Miss Peregrine's Home for Peculiar Children, Assassin's Creed

11/23/2017

imageworks/OpenShadingLanguage: Advanced shading language for production GI renderers

- (2017) / upcoming Lego Batman, The Great Wall, A Cure for Wellness, Logan, Power Rangers, Life, Smurfs: The Lost Village, The Fate of the Furious, Alien Covenant, Guardians of the Galaxy 2, The Mummy, Wonder Woman, Cars 3, Baby Driver, Spider-Man: Homecoming, Dunkirk, The Emoji Movie, Detroit, Kingsman: The Golden Circle, Lega Ninjago Movie, Blade Runner 2049, Geostorm, ...

## Building OSL

Please see the INSTALL.md file in the OSL distribution for instructions for building the OSL source code.

## Contacts, Links, and References

OSL GitHub page

Read or subscribe to the OSL development mail list

Email the lead architect: Ig AT imageworks DOT com

Most recent PDF of the OSL language specification

OSL home page at SPI

Sony Pictures Imageworks main open source page

If you want to contribute code back to the project, you'll need to sign a Contributor License Agreement.

## Credits

The original designer and project leader of OSL is Larry Gritz. Other early developers of OSL are (in order of joining the project): Cliff Stein, Chris Kulla, Alejandro Conty, Jay Reynolds, Solomon Boulos, Adam Martinez, Brecht Van Lommel.

134



11/23/2017

imageworks/OpenShadingLanguage: Advanced shading language for production GI renderers

Additionally, many others have contributed features, bug fixes, and other changes: Steve Agland, Shane Ambler, Martijn Berger, Farchad Bidgolirad, Nicholas Bishop, Stefan Büttner, Matthaus G. Chajdas, Thomas Dinges, Henri Fousse, Syoyo Fujita, Derek Haase, Sven-Hendrik Haase, John Haddon, Daniel Heckenberg, Ronan Keryell, Elvic Liang, Max Liani, Bastien Montagne, Erich Ocean, Mikko Ohtamaa, Alex Schworer, Sergey Sharybin, Stephan Steinbach, Esteban Tovagliari, Alexander von Knorring, Roman Zulak. (Listed alphabetically; if we've left anybody out, please let us know.)

We cannot possibly express sufficient gratitude to the managers at Sony Pictures Imageworks who allowed this project to proceed, supported it wholeheartedly, and permitted us to release the source, especially Rob Bredow, Brian Keeney, Barbara Ford, Rene Limberger, and Erik Strauss.

Huge thanks also go to the crack shading team at SPI, and the brave lookdev TDs and CG supes willing to use OSL on their shows. They served as our guinea pigs, inspiration, testers, and a fantastic source of feedback. And of course, the many engineers, TDs, and artists elsewhere who incorporated OSL into their products and pipelines, especially the early risk-takers at Chaos Group, Double Negative, Pixar, DNA, Isotropix, and Animal Logic. Thank you, and we hope we've been responsive to your needs.

OSL was not developed in isolation. We owe a debt to the individuals and studios who patiently read early drafts of the language specification and gave us very helpful feedback and additional ideas, as well as to the continuing contributions and feedback of its current developers and users at other VFX and animation studios.

The OSL implementation depends upon several other open source packages, all with compatible licenses:

- OpenImageIO (c) Larry Gritz, et al
- Boost - various authors
- IlmBase (c) Industrial Light & Magic
- LLVM Compiler Infrastructure

OSL's documentation incorporates parts of Markdeep (c) 2015-2016, Morgan McGuire, and highlight.js (c) 2006, Ivan Sagalaev, both distributed under BSD licenses.

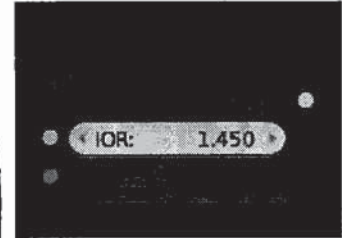
# **EXHIBIT 123**



Docs » Render » Cycles Render Engine » Nodes » Input Nodes » Fresnel Node

## Fresnel Node

The *Fresnel* or *Dielectric Fresnel* node computes how much light is reflected off a layer, where the rest will be refracted through the layer. The resulting weight can be used for layering shaders with the *Mix Shader* node. It is dependent on the angle between the surface normal and the viewing direction.



Fresnel Node.

The most common use is to mix between two BSDFs using it as a blending factor in a mix shader node. For a simple glass material you would mix between a glossy refraction and glossy reflection. At grazing angles more light will be reflected than refracted as happens in reality.

For a two-layered material with a diffuse base and a glossy coating, you can use the same setup, mixing between a diffuse and glossy BSDF. By using the Fresnel as the blending factor you are specifying that any light which is refracted through the glossy coating layer would hit the diffuse base and be reflected off that.

## Inputs

### IOR

Index of refraction (IOR) of the material being entered.

### Normal

Input meant for plugging in bump or normal maps which will affect the output.

## Properties

This node has no properties.

## Outputs

### Factor

Fresnel weight, indicating the probability with which light will reflect off the layer rather than passing through.


## **EXHIBIT 124**

## Scalar Product of Vectors

The scalar product and the vector product are the two ways of multiplying vectors which see the most application in physics and astronomy. The scalar product of two vectors can be constructed by taking the component of one vector in the direction of the other and multiplying it times the magnitude of the other vector. This can be expressed in the form:

$$\vec{A} \cdot \vec{B} = A B \cos \theta$$

(Calculation)



$\vec{A}$  denotes vector  
 $A$  denotes the magnitude of the vector.

If the vectors are expressed in terms of unit vectors  $i, j,$  and  $k$  along the  $x, y,$  and  $z$  directions, the scalar product can also be expressed in the form:

$$\vec{A} \cdot \vec{B} = A_x B_x + A_y B_y + A_z B_z \quad \text{where}$$

(Applications)

$$\vec{A} = A_x \vec{i} + A_y \vec{j} + A_z \vec{k}$$

$$\vec{B} = B_x \vec{i} + B_y \vec{j} + B_z \vec{k}$$

The scalar product is also called the "inner product" or the "dot product" in some mathematics texts.

Matrix approach to scalar product

[HyperPhysics\\*\\*\\*\\*\\* Mechanics](#)

*R Nave*

[Go Back](#)

[Index](#)

[Vector concepts](#)

## Scalar Product Calculation

You may enter values in any of the boxes below. Then click on the symbol for either the scalar product or the angle. The vectors  $A$  and  $B$  cannot be unambiguously calculated from

## **EXHIBIT 125**

11/25/2017

max - C++ Reference

This website uses cookies. By continuing, you give permission to deploy cookies, as detailed in our privacy policy.

ok

Search:

Go

Reference

<algorithm>

max

Not logged in

log in

register

C++
Information
Tutorials
Reference
Articles
Forum
Reference
C library:
Containers:
Input/Output:
Multi-threading:
Other:
<algorithm>
<bitset>
<chrono>
<codecv>
<complex>
<exception>
<functional>
<initializer_list>
<iterator>
<limits>
<locale>
<memory>
<new>
<numeric>
<random>
<ratio>
<regex>
<stdexcept>
<string>
<system_error>
<tuple>
<typeindex>
<typeinfo>
<type_traits>
<utility>
<valarray>

function template

std::max

C++98

C++11

C++14

default (1)

custom (2)

initializer list (3)

```
template <class T> constexpr const T& max (const T& a, const T& b);
template <class T, class Compare>
constexpr const T& max (const T& a, const T& b, Compare comp);
template <class T> constexpr T max (initializer_list<T> il);
template <class T, class Compare>
constexpr T max (initializer_list<T> il, Compare comp);
```

<algorithm>

Return the largest

Returns the largest of *a* and *b*. If both are equivalent, *a* is returned.

The versions for *initializer lists* (3) return the largest of all the elements in the list. Returning the first of them if these are more than one.

The function uses operator< (or *comp*, if provided) to compare the values.

The behavior of this function template (C++98) is equivalent to:

```
1 template <class T> const T& max (const T& a, const T& b) {
2   return (a<b)?b:a;      // or: return comp(a,b)?b:a; for version (2)
3 }
```

Parameters

*a*, *b*

Values to compare.

*comp*

Binary function that accepts two values of type *T* as arguments, and returns a value convertible to bool. The value returned indicates whether the element passed as first argument is considered less than the second.



# **EXHIBIT 126**

11/25/2017

min - C++ Reference

This website uses cookies. By continuing, you give permission to deploy cookies, as detailed in our privacy policy.

ok

Search:

Go

Not logged in

register

log in

Reference <algorithm> min

C++
Information
Tutorials
Reference
Articles
Forum
Reference
C library:
Containers:
Input/Output:
Multi-threading:
Other:
<algorithm>
<bitset>
<chrono>
<codecv>
<complex>
<exception>
<functional>
<initializer_list>
<iterator>
<limits>
<locale>
<memory>
<new>
<numeric>
<random>
<ratio>
<regex>
<stdexcept>
<string>
<system_error>
<tuple>
<typeindex>
<typeinfo>
<type_traits>
<utility>
<valarray>

function template

std::min

<algorithm>

C++98 C++11 C++14

default (1) template <class T> constexpr const T& min (const T& a, const T& b);  
custom (2) template <class T, class Compare>  
constexpr const T& min (const T& a, const T& b, Compare comp);  
initializer list (3) template <class T> constexpr T min (initializer\_list<T> il);  
template <class T, class Compare>  
constexpr T min (initializer\_list<T> il, Compare comp);

### Return the smallest

Returns the smallest of *a* and *b*. If both are equivalent, *a* is returned.

The versions for *initializer lists* (3) return the smallest of all the elements in the list. Returning the first of them if these are more than one.

The function uses operator< (or *comp*, if provided) to compare the values.

The behavior of this function template (C++98) is equivalent to:

```
1 template <class T> const T& min (const T& a, const T& b) {  
2   return !(b<a)?a:b; // or: return !comp(b,a)?a:b; for version (2)  
3 }
```

### Parameters

*a*, *b*

Values to compare.

*comp*

Binary function that accepts two values of type *T* as arguments, and returns a value convertible to *bool*. The value returned indicates whether the element passed as first argument is considered less than the second.

143

## **EXHIBIT 127**

# BlenderDiplom (/index.php)



## Interviews

### Interview: Larry Gritz - Lead Developer of OSL (/en/interviews/531-interview-larry-gritz-lead-developer-of-osl.html)

Written by Gottfried Hofmann

- Print (/en/interviews/531-interview-larry-gritz-lead-developer-of-osl.html?tmpl=component&print=1&layout=default)
- Email (/en/component/mailto/?tmpl=component&template=sj\_plus&link=91b2aa03e5cc21d0c3d0668f6f9df39a2e205f20)
- Published: 10 October 2013

145





Larry Gritz, lead developer of the Open Shading Language at SPI

BlenderDiplom interviews Larry Gritz, lead developer of the Open Shading Language (OSL) at Sony Pictures Imageworks. OSL has been sticking around in Cycles for a while and is supported by the latest version of V-Ray which is currently in Beta. This interview was conducted in May 2013 and was first released in German language in the magazine Digital Production (<http://www.digitalproduction.com/>).

**BD: What changes did OSL introduce to the shading workflow at Sony Pictures Imageworks?**

LG: Prior to OSL, SPI's shaders were compiled C plugins for Arnold. Our shader writers were spending more and more time dealing with low-level details -- C is rather clunky to use as a shading language, and very error-prone -- and dealing with headaches of interfacing with the renderer internals. Furthermore, it was increasingly difficult to achieve the kind of physically-based lights and materials that we wanted, and it was clear that we needed a different set of abstractions than had been used in prior shading systems.

OSL really changes three things for us at once: First, because the syntax and semantics of the language are designed with shading in mind, it is simply a more convenient notation for describing shading (especially compared to raw C). Second, it presents a really clean API that doesn't directly expose ugly or complex renderer internals. These two things put together mean that nearly all of the lines of code of a shader -- and the time and attention of the shader writer -- are about describing the material, and a great amount of work that was shuffling data around and dealing with renderer internals is eliminated. Finally, there is a big conceptual shift, from computing the appearance from one direction, which would have to include all the ugly details of sampling and integration in the shader itself, to computing view-independent material closures (which leave the sampling and integration up to the renderer, which is much better equipped to do it well). That has been a big enabler of our efforts to have more physically-based light and material models and get more pleasing and accurate images "out of the box."



**BD: What were the motivations behind open sourcing OSL?**

LG: Proprietary tools sometimes have a competitive advantage, but they also have a real cost: any tool or format that is truly unique to your facility is one where you can never hire somebody who already is an expert, you will never find external documentation, you will not have your 3rd party tools support it without a lot of additional plug-in development that you have to take on yourself.

So we were betting that any advantage of keeping it to ourselves would be small compared to the really big benefits of having an OSL ecosystem that extended beyond just SPI. We envisioned a day when 3rd party tools we use would support it, when we could hire TDs who were already familiar with this method of shading. It's even an advantage when hiring people who don't already know OSL, because it's a fact of life that artists move from facility to facility, and it's more appealing for a TD to learn skills that they feel will be valuable even after leaving Sony, than learning details about tools they'll never see again.

Also, it's very professionally satisfying for the software developers, who often don't get film credits and can't easily point to particular shots they worked on, to be working on things that are externally visible and get recognition from their peers at other VFX facilities.

Knowing that people will actually see your work makes you put more effort into making solid code you can be proud of. So we really believe that by developing OSL as an open source project, the implementation is better.

**BD: Did you get many contributions from outside yet?**

LG: Yes! Even before OSL was implemented, we circulated early drafts to some other interested studios and got lots of great early feedback even from other major well-known studios.

In terms of actual code contributions, the main feature authors are at SPI, but we have had many contributions from elsewhere. Especially now that OSL has been incorporated into Blender/Cycles, V-Ray, and Autodesk Beast, we regularly get contributions from the authors of those packages that include minor bug fixes, optimizations, and changes to the code that allow smoother porting and compilation for platforms we don't tend to use ourselves (particularly aid to getting it running nicely on Windows).

**BD: OSL was open sourced in 2010, now we have 2013 and still Blender Cycles is the only render engine on the open market that supports OSL. Are the developers of render engines hesitant to adopt it and if so, what do you think are the reasons?**

LG: That's not quite correct; it's also incorporated into Autodesk Beast and being incorporated into V-Ray, both of which have substantial user bases.

Also, although OSL has been open sourced since 2010 (in the sense that we did much of the initial development out in the open, even before it was completed), it was only mid-2012 that SPI completed the first "all-OSL" films, which was an important milestone that lots of people were waiting for before taking a gamble on it. So it's really only been one year of what you'd consider a 1.0, fully production-hardened state, and 3 major engines have incorporated OSL to date that we are aware of.

It's hard to switch an existing render engine to OSL for three reasons:

- (1) the shader execution engine (and almost certainly the texture system, along with it) is a very large portion of a renderer to replace with an external package that you don't necessarily have full control over.
- (2) "upgrading" an existing renderer to OSL may also involve a substantial overhaul of how lights and sampling work even in the non-OSL parts of the renderer; (3) products with large existing customer bases have a lot of inertia behind their user education, expertise, and shader libraries, so it's not a small task to pivot to a different way of specifying shading.

147

# **EXHIBIT 128**



11/23/2017 18 SCIENTIFIC AND TECHNICAL ACHIEVEMENTS TO BE HONORED WITH ACADEMY AWARDS | Oscars.org | Academy of Motion Picture Arts and Sciences  
Wednesday, January 4, 2017 - 12:30

The Academy of Motion Picture Arts and Sciences announced today that 18 scientific and technical achievements represented by 34 individual award recipients, as well as five organizations, will be honored at its annual Scientific and Technical Awards Presentation on Saturday, February 11, 2017 at the Beverly Wilshire in Beverly Hills.

“This year we are particularly pleased to be able to honor not only a wide range of new technologies, but also the pioneering digital cinema cameras that helped facilitate the widespread conversion to electronic image capture for motion picture production,” said Ray Feeney, Academy Award® recipient and chair of the Scientific and Technical Awards Committee. “With their outstanding, innovative work, these technologists, engineers and inventors have significantly expanded filmmakers’ creative choices for moving image storytelling.”

Unlike other Academy Awards to be presented this year, achievements receiving Scientific and Technical Awards need not have been developed and introduced during 2016. Rather, the achievements must demonstrate a proven record of contributing significant value to the process of making motion pictures.

The Academy Awards for scientific and technical achievements are:

#### TECHNICAL ACHIEVEMENT AWARDS (ACADEMY CERTIFICATES)

To **Thomson Grass Valley** for the design and engineering of the pioneering Viper FilmStream digital camera system.

*The Viper camera enabled frame-based logarithmic encoding, which provided uncompressed camera output suitable for importing into existing digital intermediate workflows.*

To **Larry Gritz** for the design, implementation and dissemination of Open Shading Language (OSL).

*OSL is a highly optimized runtime architecture and language for programmable shading and texturing that has become a de facto industry standard. It enables artists at all levels of technical proficiency to create physically plausible materials for efficient production rendering.*

To **Carl Ludwig, Eugene Troubetzkoy and Maurice van Swaaij** for the pioneering development of the CGI Studio renderer at Blue Sky Studios.

*CGI Studio’s groundbreaking ray-tracing and adaptive sampling techniques, coupled with streamlined artist controls, demonstrated the feasibility of ray-traced rendering for feature film production.*

To **Brian Whited** for the design and development of the Meander drawing system at Walt Disney Animation Studios.